第2章 Python数据处理基础

本章概要

Python程序开发语言近年来发展迅速，广泛应用于科学计算、数据挖掘和机器学习领域。数据分析是科学计算的重要组成，是必不可少的环节。Python语言不仅提供了基本的数据处理功能，还配备了很多高质量的机器学习和数据分析库。

本章介绍Python的基本数据处理功能，讲解如何利用Python进行数据控制、处理、整理和分析等操作，以及如何使用Python进行数据文件读写。随后对Python的数据分析模块进行简介，讲解NumPy和Pandas提供的数据文件访问功能。

学习目标

当完成本章的学习后，要求：

（1）了解Python程序开发环境的使用。

（2）掌握Python的基本数据类型。

（3）掌握Python读写文件的方法。

（4）熟悉使用NumPy获取数据文件内容的方法。

（5）熟悉使用Pandas存取数据文件的方法。

2.1 Python程序开发技术

Python是荷兰数学和计算机科学研究学会的Guido van Russum于1989年创建的一门编程语言。Python是开源项目，其解释器的全部代码都可以在Python的官方网站自由下载。Python的重要特点如下。

1. 面向对象

   Python既支持面向过程的编程，也支持面向对象的编程。Python支持继承、重载，代码可以重用。

2. 数据类型丰富

   Python提供了丰富的数据结构，包括列表、元组、集合、字典，此外NumPy、Pandas等资源库还提供了高级数据结构。

3. 具有功能强大的模块库

   Python是免费的开源编程语言，有许多优秀的开发者为Python开发了功能强大的拓展包，供其他人免费试用。

4. 易拓展

   Python语言的底层是由C/C++编写的，开发者可以在Python中调用C/C++编写的程序，可以极大地提高程序运行速度，同时保持程序的完整性。

5. 可移植性

因为Python是开源的，所有Python程序不依赖系统的特性，无需修改就可以在任何支持Python的平台上运行。

接下来通过下面的实例，来了解Python程序设计的基本方法。

【例2.1】 Python语言综合示例

```
import random               #包含随机数模块，以生成随机数
# 定义fib_loop函数，构造斐波那契数列
def fib_loop(n):
    listNum=[]
    a,b=0,1
    # for结构，循环体重复运行n次
    for i in range(n):
        a,b =b,a+b
        listNum.append(a)
        print(i,listNum)
    return listNum                  #返回一个数据列表listNum

listPlan = ['吃零食','学习','学习','学习','看电影','学习','旅游','睡觉','学习']
listNum = fib_loop(6)               #调用fib_loop函数生成斐波那契数列
varIdx = random.randint(0,5)        #生成0~5的随机数varIdx
varRandom = listNum[varIdx]
print('今日计划：',listPlan[varRandom])
```

运行结果：

```
0 [1]
1 [1, 1]
2 [1, 1, 2]
3 [1, 1, 2, 3]
4 [1, 1, 2, 3, 5]
5 [1, 1, 2, 3, 5, 8]
今日计划： 学习
```

程序首先定义了fib_loop函数，用来生成斐波那契数列，并在主程序中调用了fib_loop函数，生成斐波那契数列为$[1,1,2,3,5,8]$。

程序通过包含random模块，并使用`random.randint(0,5)`函数生成0~5的随机整数。然后将此随机数作为下标读取对应位置的斐波那契数，再使用该斐波那契数作为listPlan数组的下标，得到推荐事件，可以看出，每次推荐的事件均为“学习”。

思考：本例中，`random.randint(0,5)`能否改成`random.randint(0,8)`？

分析：由于斐波那契数列中低6个数是13，而本例中的listPlan中只有9个项目，如果修改为0~8的随机数，当产生的随机数大于5时，会导致非法引用，所以不能修改为`random.randint(0,8)`。

使用Python完成机器学习任务，需要熟练掌握Python编程方法，包括循环结构、选择结构、函数使用等都是较常用的基本知识。

2.2 基本数据类型

Python的标准数据类型有6种，包括Number（数字）、String（字符串）、List（列表）、Tuple（元组）、Set（集合）和Dictionary（字典）。

根据数据对象是否可变，这6种标准数据类型又可以划分为两类：可变数据类型和不可变数据类型。

可变数据类型在声明时会开辟一块内存空间，使用Python的内置方法对内存中的数据进行修改时，内存地址不会发生变化。可变数据包括列表、字典和集合。

不可变数据类型在声明时也会开辟一块内存，但不能改变这块内存中得到数据。如果改变了变量的赋值，则会重新开辟一块内存空间。不可变数据有数字、字符串、元组。

1. Number（数字）类型

Python的数字类型包括int、float、bool和complex复数类型。当指定一个值时，就创建了一个Number类型的对象。

【例2.2】 数值类型不可改变

```
i = 3
print(id(i))
i+=1
print(id(i))
```

运行结果：

```
140736415637984
140736415638016
```

通过id函数，可以看到变量i在加1后，内存地址已经改变。

2. String（字符串）类型

Python中的字符串用半角的单引号或双引号括起来，对于字符串内的特殊字符，使用反斜杠“\”进行转义。

在Python中，获取字符串的一部分的操作称为切片，截取格式为：

列表变量[头下标 : 尾下标]

正序访问时，可以获取从头下标到尾下标减1位置的字符。也可以逆序读取。

Python字符串的首字母下标为0，位置与该位置上的数值交错出现，可以把这种访问方式生动地理解为“栅栏式”访问，即每个字符的位置位于字符的前面。

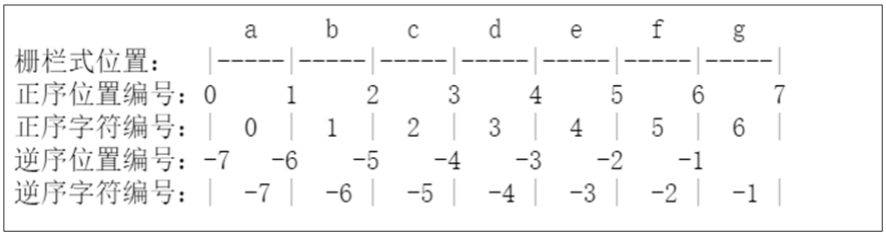

图2.1 序列访问及切片方法

【例2.3】 字符串的访问

```
str = 'Picture'
print (str[1:3])   # 第二、三个字符
print (str[-3:-1]) # 倒数第二、三个字符
print (str[3:-1])  # 正数第四个到倒数第二个字符
print (str[-6:7])  # 倒数第六个到正数第七个字符
print (str[2:])    # 第三个字符开始的所有字符
print (str * 2)    # 输出字符串两次
print (str + "TEST") # 连接字符串
```

运行结果：

```
ic
ur
tur
icture
cture
PicturePicture
PictureTEST
```

由于字符串是不可变类型，所以向字符串某位置赋值会导致错误。

【例2.4】 字符串的赋值

```
word = 'Python'
print(word[0], word[5])
print(word[-1], word[-6])
```

运行结果：

```
P n
n P
```

如果继续添加一行语句：

```
word[0] = 'Q'
```

由于无法修改 word字符串，因此会导致错误：TypeError: 'str' object does not support item assignment

如果需要修改字符串的内容，可以使用重新赋值语句，即生成一个新的word变量。

3. List（列表）类型

List（列表）使用方括号[]进行定义，数据项之间用逗号分隔。列表是 Python 中使用最频繁的数据类型。列表的数据项可以是数字、字符串，也可以是列表。

列表截取语法与字符串类似，格式如下：

列表变量[头下标 : 尾下标]

正序访问时，索引值从0开始，截取从头下标到尾下标减1位置的元素；如果是逆序访问，则-1是末尾位置。

【例2.5】 列表的访问

```
list = [ 'a', 56 , 1.13, 'HelloWorld',[7,8,9] ]
print(list)       #完整列表
print(list[4])    # 第五个元素
print(list[-2:5]) # 从倒数第二个到正数第五个元素
print(list[2:])   # 第三个元素开始的所有元素
```

运行结果：

```
['a', 56, 1.13, 'HelloWorld', [7, 8, 9]]
[7, 8, 9]
['HelloWorld', [7, 8, 9]]
[1.13, 'HelloWorld', [7, 8, 9]]
```

与字符串不一样的是，列表中的元素可以赋值修改。List还内置了很多方法，例如append()、pop()等等。

【例2.6】 列表元素的修改

```
a = [1, 2, 3, 4, 5, 6]
a[0] = 9    # 将第一个元素设为9
print(a)
a.append(7) # 在列表末尾追加7
print(a)
a[2:5] = [] # 将第三到五个元素值设置为空值
print(a)
a.pop(2)    # 将第三个元素移除
print(a)
```

```
[9, 2, 3, 4, 5, 6]
[9, 2, 3, 4, 5, 6, 7]
[9, 2, 6, 7]
[9, 2, 7]
```

在实际应用中，经常需要对列表中的数据项进行遍历（也称为**迭代**）。Python中常用的列表迭代方法有三种：**（1）for循环遍历；（2）按索引序列遍历；（3）按下标遍历。**其中，按索引遍历一般使用enumerate()函数，将可遍历的数据对象（如列表、元组和字符串）组合为一个索引序列，同时列出数据和数据下标，再结合for循环进行遍历。

【例2.7】 列表的遍历

```
lis= ['蚂蚱','螳螂','蝈蝈','蝗虫','蛐蛐']
#(1)直接遍历
for item in lis:
    print(item)
 #(2)按索引遍历
for i in enumerate(lis):
    print(i)
#(3)对于列表类型，还有一种通过下标遍历的方式，如使用range()函数
for i in range(len(lis)):
    print(lis[i])
```

运行结果:

```
蚂蚱
螳螂
蝈蝈
蝗虫
蛐蛐

(0, '蚂蚱')
(1, '螳螂')
(2, '蝈蝈')
(3, '蝗虫')
(4, '蛐蛐')

蚂蚱
螳螂
蝈蝈
蝗虫
蛐蛐
```

4. Tuple（元组）类型

元组写在小括号（）中，元素之间用逗号分隔，元素可以具有不同的类型。元组（Tuple）与列表类似，但元组的元素不能修改。

元组的截取方式与字符串和列表都类似，下标从0开始，末尾的位置从-1开始。

【例2.8】 元组的访问

```
tuple = ('SpiderMan',2017,33.4,'Homecoming',14)
tinytuple = (16,'Marvel')
print(tuple)                 #输出完整元组
print(tuple[0])              #输出元组第一个元素
print(tuple[3:4])            #输出元组第四个元素
print(tuple+tinytuple)
```

```
('SpiderMan', 2017, 33.4, 'Homecoming', 14)
SpiderMan
('Homecoming',)
('SpiderMan', 2017, 33.4, 'Homecoming', 14, 16, 'Marvel')
```

虽然元组的元素不可改变，但如果元组内部的数据项是可变的类型，则该数据项可以修改。

【例2.9】 修改元组中的List类型数据项

```
tuple = ([16,'Marvel'],'SpiderMan',2017,33.4,'Homecoming',14)
print(tuple[0])
tuple[0][0]='Marvel'
tuple[0][1]='16'
print(tuple)
```

```
[16, 'Marvel']
(['Marvel', '16'], 'SpiderMan', 2017, 33.4, 'Homecoming', 14)
```

5. Dictionary（字典）类型

字典是一种可变容器模型，且可存储任意类型对象。字典使用大括号{ }定义，格式如下。

d = {key1:value1,key2:value2}

字典的每个键值（key/value）对用冒号分隔，键值对之间用逗号分隔。键一般是唯一的，如果出现了重复，则后面的键值对会替换前面的键值对。值的数据及类型不限，可以是字符串、数字或元组。

1）字典的访问

访问字典中的值需要使用字典的键值，这个键值用方括号括起来，格式如下：

dt['key']

【例2.10】 字典的访问

```
dict = {'Name':'Mary','Age':7,'Class':'First'};
print(dict)
print('Name:',dict['Name'])
print('Age:',dict['Age'])
```

```
{'Name': 'Mary', 'Age': 7, 'Class': 'First'}
Name: Mary
Age: 7
```

【例2.11】 列表可以作为字典的value值

```
dict = {'Name':['Mary','Tom','Philp'],'Age':[7,8,9],'Class':
['1st','2nd','3rd']}
print(dict)
print('Name:',dict['Name'])
dict['Age']=[8,9,10]
print('Age:',dict['Age'])
```

```
{'Name': ['Mary', 'Tom', 'Philp'], 'Age': [7, 8, 9], 'Class': ['1st',
'2nd', '3rd']}
Name: ['Mary', 'Tom', 'Philp']
Age: [8, 9, 10]
```

2）修改字典

可以向字典添加键/值对，也可以修改或删除字典的键/值对。

【例2.12】 修改字典

```
dict = {'Name':'Zara','Class':'First'}
# 添加
dict['Gender'] = 'Female'
print(dict)
# 修改update
dict.update({'No':'001'})
print(dict)
# 也可以用update方法添加/修改多个数据
dict.update({'Gender':'F','ID':'1'})
print(dict)
```

```
{'Name': 'Zara', 'Class': 'First', 'Gender': 'Female'}
{'Name': 'Zara', 'Class': 'First', 'Gender': 'Female', 'No': '001'}
{'Name': 'Zara', 'Class': 'First', 'Gender': 'F', 'No': '001', 'ID': '1'}
```

删除一个字典键值对用del命令，清空字典用clear命令。

【例2.13】 删除字典元素

```
del dict['Gender']
print(dict)
dict.clear()
print(dict)
```

```
{'Name': 'Zara', 'Class': 'First', 'No': '001', 'ID': '1'}
{}
```

6. Set（集合）类型

集合由一列无序的、不重复的数据项组成。Python中的集合是可变类型。与数学中的集合概念相同，集合中每个元素都是唯一的。同时，集合不设置顺序，每次输出时元素的排序可能都不相同。

集合使用大括号，形式上和字典类似，但数据项不是成对的。

1）创建集合

创建集合可以使用大括号{}或者set()函数。但创建一个空集合必须使用set()函数而不能用{}，因为空的大括号{}创建的是空的字典。要建立一个由$(v_1, v_2, \cdots)$组成的集合mySet，可以使用$mySet = \{v_1, v_2, \cdots\}$。

还可以使用List列表来创建集合，此时列表中的数据项直接作为集合的元素。生成的集合和原List列表相比，数据项顺序有可能不同，并且会去除重复数据项。

例如，要由列表myList建立一个名为mySet的集合，可以使用mySet=set(myList)。

【例2.14】 创建集合

```
# 创建一个空集合
var = set()
print(var,type(var))  #显示集合内容和类型
# 具有数据的集合
var = {'LiLei','HanMeiMei','ZhangHua','LiLei','LiLei'}
print(var,type(var))
```

```
set() <class 'set'>
{'LiLei', 'ZhangHua', 'HanMeiMei'} <class 'set'>
```

【例2.15】 集合成员检测

```
#判断元素在集合内
result1 = 'LiLei' in var
result2 = 'Lilei' in var     #大小写敏感
print(result1,result2)
#判断元素不在集合内
result = 'LiLei' not in var
print(result)
```

```
True False
False
```

2）添加、删除集合元素

为集合添加数据项有两种常用方法：add()和update()。删除集合项的常用方法是remove()

【例2.16】 添加、删除集合元素

```
var = {'LiLei','HanMeiMei','ZhangHua'}
var.add('LiBai')    #add方法添加元素
print(var)
var.update('DuFu')  #update方法首先拆分元素，然后依次添加
print(var)
var.remove('D')
print(var)
```

```
{'ZhangHua', 'LiBai', 'HanMeiMei', 'LiLei'}
{'u', 'ZhangHua', 'HanMeiMei', 'LiLei', 'LiBai', 'D', 'F'}
{'u', 'ZhangHua', 'HanMeiMei', 'LiLei', 'LiBai', 'F'}
```

3）集合的遍历

集合中的元素可以使用遍历进行访问。可以使用直接遍历，也可以使用enumerate索引进行遍历。但是，集合类型不支持range()方式的遍历。

【例2.17】 添加、删除集合元素

有一个集合anml，其内容为{'紫貂','松貂','青鼬','狼獾'}，对anml集合进行遍历

方法一：

```
for item in anml:
    print(item)
```

```
狼獾
松貂
青鼬
紫貂
```

方法二：

```
for item in enumerate(anml):
    print(item)
```

```
0, '青鼬')
(1, '松貂')
(2, '紫貂')
(3, '狼獾')

Process finished with exit code 0
```

range()方式不支持：

```
for item in range(len(anml)):
    print(anml[item])
```

```
TypeError: 'set' object is not subscriptable
```

修改：

```
anml = ['紫貂','松貂','青鼬','狼獾']    #列表的遍历
print(type(anml))
for item in range(len(anml)):
    print(anml[item])
```

```
<class 'list'>
紫貂
松貂
青鼬
狼獾
```

```
anml = ('紫貂','松貂','青鼬','狼獾')
print(type(anml))
for item in range(len(anml)):
    print(anml[item])
```

```
<class 'tuple'>
紫貂
松貂
青鼬
狼獾
```

4）Python集合操作符号

Python集合类型与数学中的集合类似，支持集合的交集、并集、差集、包含等运算。常见数学集合运算符与Python结合操作符的对比如表2.1所示。

表2.1 数学集合运算符与Python集合操作符对比

集合操作	数学集合运算符	Python集合操作符
差集	−	−
交集	∩	&
并集	∪	|
不等于	≠	!=
等于	=	==
包含于	∈	in
不包含于	∉	not in

【例2.18】 集合的交集、并集、差集

```
# 分别构造獾和貂的两个集合
Huan = {'猪獾','蜜獾','狼獾'}
Diao = {'紫貂','松貂','美洲水鼬','狼獾'}
# 交集
Diaoxiong = Huan & Diao
print('貂熊是：',Diaoxiong)
# 并集
YouKe = Huan | Diao
print('鼬科是：',YouKe)
# 差集
DiaoT = Diao - Huan
print('除去獾的貂类：',DiaoT)
```

```
貂熊是： {'狼獾'}
鼬科是： {'猪獾', '松貂', '紫貂', '蜜獾', '狼獾', '美洲水鼬'}
除去獾的貂类： {'松貂', '紫貂', '美洲水鼬'}
```

2.3 数据文件读写

机器学习的本质是数据处理，以及在此基础上的算法运行。如果数据是少量的，临时的，可以使用标准数据类型变量进行存储。不过在实际应用中，经常使用大量的数据，这时需要使用数据文件。

2.3.1 打开与关闭文件

Python提供了标准的文件操作功能，可以对文件进行读写操作。

1. 打开文件

打开文件的内置函数是open()函数，打开文件后会创建一个文件对象。对文件的访问通过这个文件对象进行。

语法：

open(file_name [, access_mode][, buffering])

主要参数说明：

file_name：字符串类型，要访问的文件名称。

access_mode：文件的打开模式，如读取、写入或追加等。可选参数，默认为r（只读模式）。写数据常用的是'w'和'a'模式，分别表示改写和添加。

buffering：表示文件缓冲区的策略，可选。当值为0时，表示不使用缓冲区。

例如：f = open('datafile.txt','w')

2. 写入文件

向文件中写入数据，使用文件对象的write()方法，参数为要写入文件的字符串。

例如，f.write('some data')

3. 关闭文件

关闭文件使用文件对象的close()方法。

例如，f.close()

【例2.19】 打开文件并写入数据

```
filename = 'INFO.txt'
f = open(filename,'w')    #清空原文件数据，若文件不存在则创建新文件
f.write('I am Zhangsan.\n')
f.write('I am now studying in ZUST.\n')
f.close()
```

运行后，程序在当前目录生成一个INFO.txt文件，内容为写入的两行数据。

文件的读写可能会产生错误。例如，读取一个不存在的文件或者没有正常关闭的文件，会产生IOError错误。为了避免此类问题，可以使用try...finally语句，不过更方便的是使用Python的with语句。使用with语句打开文件时，不必调用f.close()方法就能自动关闭文件。即使文件读取出错，也会保证关闭文件。使用with语句访问文件，代码更简洁，能获得更好的异常处理。

【例2.20】 使用with语句打开文件

```
with open('INFO.txt','a') as f:          #'a'表示添加数据，不清除原数据
    f.write('I major in Computer Vision.\n')
```

2.3.2 读取文件内容

文件对象也提供了读取文件的方法，包括read()、readline()、readlines()等，其功能介绍如下：

1. file.read([count])

读文件，默认读整个文件。如果设置了参数count，则读取count字符，返回值为字符串。

2. file.readline()

从当前位置开始，读取文件中的一行，返回值为字符串。

3. file.readlines()

从当前位置开始，读取文件的所有行，返回值为列表，每行为列表的一项。

读取文件时，可以使用for循环对文件对象进行遍历。在实际使用时，可以根据需要选择合适的文件读取方式。

【例2.21】 read()函数读取整个文件

```
with open('INFO.txt') as f:
    contents = f.read()  #从当前位置开始读取全部内容
    print(contents)
```

```
I am Zhangsan.
I am now studying in ZUST.
I major in Computer Vision.
```

```
with open('INFO.txt') as f:
    ct10 = f.read(5)    #读5个字符
    print(ct10)
    print('========')
    contents = f.read()  #从当前位置开始读取全部内容
    print(contents)
```

```
I am
========
Zhangsan.
I am now studying in ZUST.
I major in Computer Vision.
```

有时读取的数据带有特殊字符或需要去掉的空格，如\n（换行）、\n（回车）、\t（制表符）、''（空格）等，可以使用Python提供的函数去除头尾不需要的字符。常用的去空白符（包括\n、\n、\t、''）函数如下：

strip()：去除头、尾的字符和空白符

lstrip()：去除开头的字符和空白符

rstrip()：去除开头的字符和空白符

【例2.22】 readline()函数逐行读取

```
with open('data.txt') as f:
    line1 = f.readline()    #读取第一行数据，此时已经指向第一行末尾
    line2 = f.readline()    #从上一次读取末尾开始读取（第二行）
    print(line1)
    print('========')
    print(line2)
    print('========')
    print(line1.strip())
    print('========')
    print(line2.strip())
    print('========')
    print(line1.split(),type(line1.split()))
```

```
1.5 40 thin

========
1.5 50 fat

========
1.5 40 thin
========
1.5 50 fat
========
['1.5', '40', 'thin'] <class 'list'>
```

【例2.23】 readlines()函数一次读取多行

```
with open('data.txt') as f:
    lines = f.readlines()       #将文件数据读到一个列表，每个列表项对应一行
print(lines)                    #每行数据都包含换行符
print('=================')
for line in lines:
    print(line.rstrip())
```

```
['1.5 40 thin\n', '1.5 50 fat\n', '1.5 60 fat\n', '1.6 40 thin\n', '1.6 50
thin\n', '1.6 60 fat\n', '1.6 70 fat\n', '1.7 50 thin\n', '1.7 60 thin\n',
'1.7 70 fat\n', '1.7 80 fat\n', '1.8 60 thin\n', '1.8 70 thin\n', '1.8 80
fat\n', '1.8 90 fat\n', '1.9 80 thin\n', '1.9 90 fat']
=================
1.5 40 thin
1.5 50 fat
1.5 60 fat
1.6 40 thin
1.6 50 thin
1.6 60 fat
1.6 70 fat
1.7 50 thin
1.7 60 thin
1.7 70 fat
1.7 80 fat
1.8 60 thin
1.8 70 thin
1.8 80 fat
1.8 90 fat
1.9 80 thin
1.9 90 fat
```

【例2.24】 使用for循环逐行读取文件

```
with open('data.txt') as f:
    for lineData in f:
        print(lineData.rstrip())   #去掉每行末尾的换行符
```

```
1.5 40 thin
1.5 50 fat
1.5 60 fat
1.6 40 thin
1.6 50 thin
1.6 60 fat
1.6 70 fat
1.7 50 thin
1.7 60 thin
1.7 70 fat
1.7 80 fat
1.8 60 thin
1.8 70 thin
1.8 80 fat
```

```
1.8 90 fat
1.9 80 thin
1.9 90 fat
```

2.3.3 将数据写入文件

如果需要对文件写入数据，打开方式需要选择'w'（写入）或者'a'（追加）模式。写入文件可以使用Python提供的write方法。write方法的语法如下：

fileObject.write(byte)

其中，参数byte为待写入的字符串或者字节。

【例2.25】 新建文本文件并写入内容

```
filename = 'write_data.txt'
with open(filename,'w') as f:          #'w'表示写数据，会清空原文件
    f.write('I am Zhangsan.\n')
    f.write('I am now studying in ZUST.\n')
```

【例2.26】 向文件中追加数据

```
with open(filename,'a') as f:          #'a'表示追加数据，不清楚原数据
    f.write('I major in Computer Vision.\n')
```

2.3.4 Pandas存取文件

Pandas是一个强大的分析结构化数据的工具集，基础是NumPy。本节学习Pandas模块以及它所提供的用于数据分析的基础功能。Pandas的核心功能是数据计算和处理，对外部文件读写数据也是Pandas功能的一部分。而且，可以使用Pandas在数据读写阶段对数据做一定的预处理，为接下来的数据分析做准备。

数据获取对Pandas的数据分析来说非常重要。Pandas模块提供了专门的文件输入输出函数，大致可分为读取函数和写入函数两类，如表2.2所示。

表2.2 Pandas主要读取和写入函数

读取函数	写入函数	函数功能
read_csv()	to_csv()	将csv文件读入DataFrame，默认以逗号分隔
read_excel()	to_excel()	将excel文件读取到Pandas DataFrame中
read_sql()	to_sql()	将SQL查询或数据库表读取到DataFrame中
read_json()	to_json()	读写json格式文件和字符串
read_html()	to_html()	可以读写HTML字符串/文件/URL

1. read_csv()函数

函数功能：从文件、URL、文件对象中加载带有分隔符的数据，默认分隔符是逗号。TXT文件和CSV文件可以通过Pandas中的read_csv()函数进行读取。有时候可以使用read_table()函数读取表格数据，其默认分隔符为制表符（"\t"）。read_csv()和read_table()函数都有丰富的参数可以设置。

read_csv()的格式如下：

pd.read_csv(filepath_or_buffer,sep,header,encoding,index_col,columns...)

该函数有二十多个参数，其主要参数如下：

filepath_or_buffer：字符串型，代表文件名或数据对象的路径，也可以是URL

sep：字符串型，数据的分隔符。read_csv()中默认是逗号，read_table()中默认是制表符。

header：整型或整数列表，表示此行的数据是关键字，数据从下一行开始。header默认为0，即文件第1行数据是关键字。如果文件中的数据没有关键字，需要将header设置为None。

encoding：字符串型，可选参数，注明数据的编码格式，默认为utf-8

index_col：整数，默认是None，指定行索引的列号。

【例2.27】 read_csv()读取有标题的数据

```
import pandas as pd
data1 = pd.read_csv('dataH.txt')
print(data1)
print('------------------')
data2 = pd.read_csv('dataH.txt',sep=' ') #指明分隔符
print(data2)
```

```
       Ht Wt Rt
0   1.5 40 thin
1    1.5 50 fat
2    1.5 60 fat
3   1.6 40 thin
4   1.6 50 thin
5    1.6 60 fat
6    1.6 70 fat
7   1.7 50 thin
8   1.7 60 thin
9    1.7 70 fat
10   1.7 80 fat
11  1.8 60 thin
12  1.8 70 thin
13   1.8 80 fat
14   1.8 90 fat
15  1.9 80 thin
16   1.9 90 fat
------------------
     Ht  Wt    Rt
0   1.5  40  thin
1   1.5  50   fat
2   1.5  60   fat
```

```
3   1.6  40  thin
4   1.6  50  thin
5   1.6  60   fat
6   1.6  70   fat
7   1.7  50  thin
8   1.7  60  thin
9   1.7  70   fat
10  1.7  80   fat
11  1.8  60  thin
12  1.8  70  thin
13  1.8  80   fat
14  1.8  90   fat
15  1.9  80  thin
16  1.9  90   fat
```

【例2.28】 read_csv()读取无标题的数据

```
import pandas as pd
data3 = pd.read_csv('data.txt',sep=' ')
print(data3)
print('------------------')
data4 = pd.read_csv('data.txt',sep=' ',header=None) #指明分隔符
print(data4)
```

```
    1.5  40  thin
0   1.5  50   fat
1   1.5  60   fat
2   1.6  40  thin
3   1.6  50  thin
4   1.6  60   fat
5   1.6  70   fat
6   1.7  50  thin
7   1.7  60  thin
8   1.7  70   fat
9   1.7  80   fat
10  1.8  60  thin
11  1.8  70  thin
12  1.8  80   fat
13  1.8  90   fat
14  1.9  80  thin
15  1.9  90   fat
------------------
      0   1     2
0   1.5  40  thin
1   1.5  50   fat
2   1.5  60   fat
3   1.6  40  thin
4   1.6  50  thin
5   1.6  60   fat
6   1.6  70   fat
7   1.7  50  thin
```

```
8   1.7  60  thin
9   1.7  70   fat
10  1.7  80   fat
11  1.8  60  thin
12  1.8  70  thin
13  1.8  80   fat
14  1.8  90   fat
15  1.9  80  thin
16  1.9  90   fat
```

【例2.29】 读取无标题数据并设置标题名

```
import pandas as pd
data5 = pd.read_table('data.txt',sep=' ',header=None,names=['H','W','C'])
print(data5)
```

```
      H   W     C
0   1.5  40  thin
1   1.5  50   fat
2   1.5  60   fat
3   1.6  40  thin
4   1.6  50  thin
5   1.6  60   fat
6   1.6  70   fat
7   1.7  50  thin
8   1.7  60  thin
9   1.7  70   fat
10  1.7  80   fat
11  1.8  60  thin
12  1.8  70  thin
13  1.8  80   fat
14  1.8  90   fat
15  1.9  80  thin
16  1.9  90   fat
```

2. to_csv()函数

如果需要将数据写入CSV文件，可以使用Pandas提供的to_csv()函数。使用Pandas读写的数据都是基于Pandas内置的DataFrame格式，方便继续对数据进行处理。

【例2.30】 将DataFrame格式数据得到两列写入文件

```
data5.to_csv('HW.csv',columns=['H','W'])
```

程序生成CSV文件，内容如图所示。

	A	B	C	D
1		H	W	
2	0	1.5	40	
3	1	1.5	50	
4	2	1.5	60	
5	3	1.6	40	
6	4	1.6	50	
7	5	1.6	60	
8	6	1.6	70	
9	7	1.7	50	
10	8	1.7	60	
11	9	1.7	70	
12	10	1.7	80	
13	11	1.8	60	
14	12	1.8	70	
15	13	1.8	80	
16	14	1.8	90	
17	15	1.9	80	
18	16	1.9	90	

2.3.5 NumPy存取文件

除了Pandas，NumPy也可以非常方便地存取文件，它提供了如表2.3所列地函数。

表2.3 NumPy主要读、写函数

读函数	写函数	功能
fromfile()	tofile()	存取二进制格式文件
load()	save()	存取NumPy专用地二进制格式文件
loadtxt()	savetxt()	存取文本文件，也可以访问CSV文件

可以通过loadtxt()从文本文件中读取数据，或用savetxt()把数组写入文本文件。这两个函数可以存取文本文件，也可以访问csv文件。它们在存取过程中，使用的是NumPy内置的一维和二维数组格式。

【例2.31】 使用loadtxt()读取文件

```
import numpy as np
#采用字符串数组读取文件
tmp = np.loadtxt('data.txt',dtype=np.str,delimiter=' ')
print(tmp)
print('--------------------')
tmp1 = np.loadtxt('data.txt',dtype=np.str,usecols=(1,2)) #usecols：选取数据的列
print(tmp1)
```

```
[['1.5' '40' 'thin']
 ['1.5' '50' 'fat']
 ['1.5' '60' 'fat']
 ['1.6' '40' 'thin']
 ['1.6' '50' 'thin']
 ['1.6' '60' 'fat']
 ['1.6' '70' 'fat']
 ['1.7' '50' 'thin']
```

```
 ['1.7' '60' 'thin']
 ['1.7' '70' 'fat']
 ['1.7' '80' 'fat']
 ['1.8' '60' 'thin']
 ['1.8' '70' 'thin']
 ['1.8' '80' 'fat']
 ['1.8' '90' 'fat']
 ['1.9' '80' 'thin']
 ['1.9' '90' 'fat']]
 --------------------
[['40' 'thin']
 ['50' 'fat']
 ['60' 'fat']
 ['40' 'thin']
 ['50' 'thin']
 ['60' 'fat']
 ['70' 'fat']
 ['50' 'thin']
 ['60' 'thin']
 ['70' 'fat']
 ['80' 'fat']
 ['60' 'thin']
 ['70' 'thin']
 ['80' 'fat']
 ['90' 'fat']
 ['80' 'thin']
 ['90' 'fat']]
```

【例2.32】 使用savetxt()写入数据

```
import numpy as np
x=y=z=np.arange(0,50,4.5)
# 把x数组保留一位小数写入文件
np.savetxt('X.txt',x,delimiter=',',fmt='%5.1f')
#把x数组保留三位小数写入文件
np.savetxt('X2.txt',x,delimiter=',',fmt='%7.3f')
#把三个数组按照原格式写入文件
np.savetxt('XYZ.txt',(x,y,z))
```

习题

一、选择题

1、以下不属于Python标准数据类型的是（ ）

A. DataFrame　　B. 字符串　C. 数值　D. 列表

2、使用小括号定义的数据类型是（ ）

A. 列表　　B. 集合　C. 字典　　D. 元组

3、使用{ }定义的数据类型是（ ）

A. 字典　　B. 集合　C. 列表　D. 字典或集合

4、以下关于字典中的键值的说法，正确的是（　）

A. 键值不可修改　　B. 键值不能重复　C. 键值必须是字符串　D. 以上都不对

5、以下描述中，属于集合特点的是（　）

A. 集合中的数据是无序的　　B. 集合中的数据是可以重复的

C. 集合中的数据是严格有序的　　D.集合中必须嵌套一个子集合

二、操作题

1、Python基本数据练习。

（1）建立字符串“the National Day”，取出单词“Nation”并显示

（2）建立列表["the","National","Day"]，取出单词“National”并显示。

（3）建立一个元组tpl=(['10.1','is','the'], 'National','Day')，并把值“10.1”变成"Today"，显示整个元组

（4）建立酒店客流数据字典，数据如下。Hotel = {"name":"J Hotel", "count":35, "price":162}。然后把count值修改为36.

（5）建立酒店名称的集合。Htls = {"A Hotel","B Hotel","C Hotel"}，然后检查”E Hotel“是否在集合中

2、Python文件访问练习。

编程读取数据文件fruit_data_with_colors.txt前10行数据并显示。